UNIVERSITÀ DI PISA

Corso di Laurea in Informatica Umanistica

RELAZIONE

Simulatore di stati tramite big data in sicurezza informatica

Candidato: *Samson Waldmann*

Relatore: *Fabrizio Baiardi*

Correlatore: *Mirko Tavosanis*

Anno Accademico 2017-2018

Indice

1 Introduzione

Un *Intrusion Detection System* supervisiona una rete o un dispositivo locale, rilevando attività anomale, in particolare attacchi informatici. Un attacco consiste nell'ottenimento da parte dell'attaccante di diritti, l'insieme di uno o più diritti del sistema prende il nome di stato.
Un *IDS* basato su basi di dati ha a disposizione sequenze di attacchi conosciuti.
I sensori monitorano lo stato del sistema. L'obbiettivo é di confrontare stati osservati tramite sensori con le informazioni contenute nel database e di valutare se si tratta di una violazione o meno, e nel caso lo sia, quanto é probabile e quanto tempo l'attaccante impiega mediamente per eseguirla.

Questa implementazione in particolare è focalizzata nel valutare la possibilità, dati due stati, di poter arrivare da uno stato ad un altro.

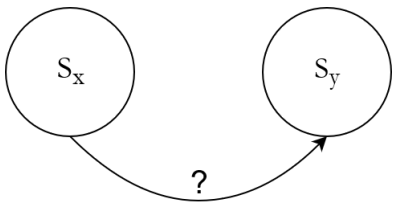


Fig. 1: Raggiungibilità di uno stato da un altro

Inoltre si prevede anche la possibilità di elencare tutti i diritti raggiungibili da uno stato dato.

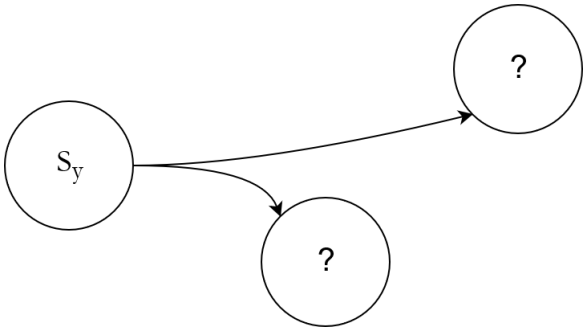


Fig. 2: Diritti raggiungibili da uno stato

L'utente ha la possibilità tramite questo programma di compiere le seguenti azioni:

- Attivando i sensori, studiare una possibile evoluzione futura dello stato degli elementi osservati.
- Simulare, componendo uno stato a piacere, la sua evoluzione futura e/o verificare se esso sia corrispondente ad un attacco noto, la sua probabilità e i relativi tempi.

1.1 Definizioni

Definiamo **DIRITTI** un insieme di coppie $< oggetto, operazione >$, ad esempio modifiche di attributi di file come permessi di scrittura.

$$diritto = < oggetto, operazione >$$

ad esempio quelli di un file unix eseguibile:

$$FILE, ATTRIB = 775$$

Definiamo **ATTACCO** l'ottenimento di *diritti* e un tempo t, il momento dell'attacco.

$$attacco = (diritti, t)$$

Definiamo **STATO** di un sistema, un insieme di diritti.

$$stato = diritti$$

ad esempio quelli ottenuti in un attacco:

$$stato = diritti(attacco)$$

o da una serie di attacchi:

$$stato = diritti(attacco_1) \cup diritti(attacco_2)$$

1.2 Dati

Una tabella che in ogni tupla contiene una sequenza di attacchi e una probabilità associata alla sequenza

att1	att2	att3	att4	p1
att1	att2	att3	att5	p2
...	...	...		
...				

In input viene fornito uno stato osservato dai sensori:

$$Input = < Stato >$$

Il programma verifica se all'interno dei dati esiste una o più corrispondenze di tale stato e, se esistono ne restituisce il tempo e la probabilità. Nel caso di più corrispondenze viene visualizzato il tempo minore e quello maggiore. Restituisce inoltre il numero di stati corrispondenti totali:

$$Output = \ stati\ corrispondenti\ totali, \begin{cases} stato(min(tempo)), probabilita \\ stato(max(tempo)), probabilita \end{cases}$$

Deve inoltre essere possibile, una volta trovata la corrispondenza di uno stato, poter verificare se sia possibile, a partire da esso attraverso altri attacchi, raggiungere ulteriori diritti.

$$Output = \{ulteriori\ diritti\ ottenibili\}$$

1.3 Algoritmo

L'obiettivo è di organizzare i dati disponibili sotto forma di grafo diretto aciclico e di applicare successivamente su di esso gli algoritmi per la ricerca di percorsi. Il singolo stato viene rappresentato come vertice. Ogni vertice ha un nome univoco e come attributi i diritti, la probabilità e il tempo.

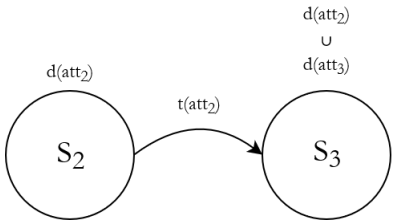
$$vertice =< Stato, probabilita >$$

Nel caso in cui due attacchi siano consecutivi in una tupla, tra i vertici degli stati che ne derivano, viene creato un arco diretto.

...	att2	att3	...	p
...	..			

$$V = \{S_2 :< diritti(att2), p >, S_3 :< diritti(att2) \cup diritti(att3), p >\}$$

$$E = \{\{S_2, S_3\}\}$$



Nel caso di sovrapposizioni, attacchi identici ma appartenenti a tuple con probabilità differenti, l'attributo probabilità viene sommato.

att3	att4	...	..	p1
att3	att4	...	..	p2

$$V = \{S_4 :< diritti(att3) \cup diritti(att4), p1 + p2 >\}$$

Un esempio completo:

att2	att3	att4	att6	p1
att2	att3	att5	att6	p2

$$V = \{S_2 :< d(att_2), p1 + p2 >,$$
$$S_3 :< d(att_2) \cup d(att_3), p1 + p2 + p(S_2) >,$$
$$S_4 :< d(att_2) \cup d(att_3) \cup d(att_4), p1 + p(S_3) + p(S_2) >,$$
$$S_5 :< d(att_2) \cup d(att_3) \cup d(att_5), p2 + p(S_3) + p(S_2) >,$$
$$S_6 :< d(att_2) \cup d(att_3) \cup (d(att_4) \vee d(att_5)) \cup d(att_6), p1 + p2 + (p(S_4) \vee p(S_5)) + p(S_3) + p(S_2) >\}$$

$$E = \{\{S_2, S_3\}, \{S_3, S_4\}, \{S_3, S_5\}, \{S_4, S_6\}, \{S_5, S_6\}\}$$

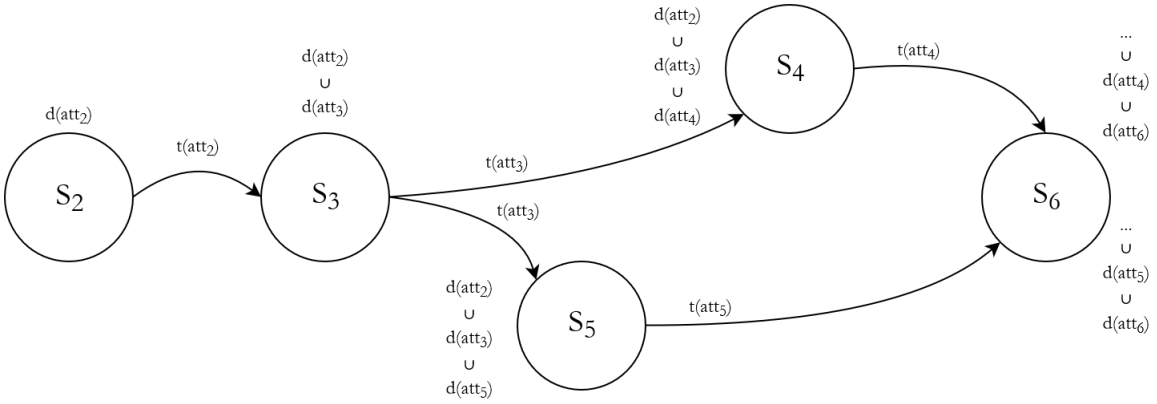


Fig. 3: I vari percorsi per raggiungere uno stato

In particolare la ricerca sul grafo avviene utilizzando il **Breadth First Traversal**.
L'algoritmo cerca tutti i percorsi possibili da un vertice **A** verso un vertice **B** verificando che tali vertici siano connessi. Nell'esempio sopra una ricerca Breadth First Traversal tra i due estremi del grafo restituirebbe due percorsi:

$$Percorso = BreadthFirstTraversal(S_2, S_6) = \begin{cases} \{S_2 \cup S_3 \cup S_4 \cup S_6\} \\ \{S_2 \cup S_3 \cup S_5 \cup S_6\} \end{cases}$$

In particolare S_6 avrà due possibili derivazioni diverse a seconda delle scelte compiute dall'attaccante:

$$S_6 = \begin{cases} \{d(att_2) \cup d(att_3) \cup d(att_4) \cup d(att_6)\} \\ \{d(att_2) \cup d(att_3) \cup d(att_5) \cup d(att_6)\} \end{cases}$$

1.3.1 Cercare uno stato

Nel nostro caso non è sufficiente trovare dei percorsi qualsiasi tra due vertici connessi.
Per scoprire se da S_x é raggiungibile S_y attraverso uno stato S_z , si può adattare la funzione Breadth First Traversal affinche durante il calcolo del percorso per S_y passi necessariamente per S_z.
Prendendo come input il seguente stato:

$$Input = < d(att_2) \cup d(att_4) \cup d(att_6) >$$

Si chiede se é possibile che l'attaccante ottenga i diritti di att_2, poi att_4 e infine att_6.

Gli stati intermedi formano dei vincoli portando l'algoritmo a scartare percorsi che non li includono.
La nuova funzione **BreadthFirstTraversalConVincolo** prende in argomento l'input completo e non più lo stato iniziale e quello cercato.
Considerando l'esempio precedente, soltanto una delle due soluzione risponde ai requisiti:

$$Percorso = BreadthFirstTraversalConVincolo(Input) = \{S_2, S_3, S_4, S_6\}$$

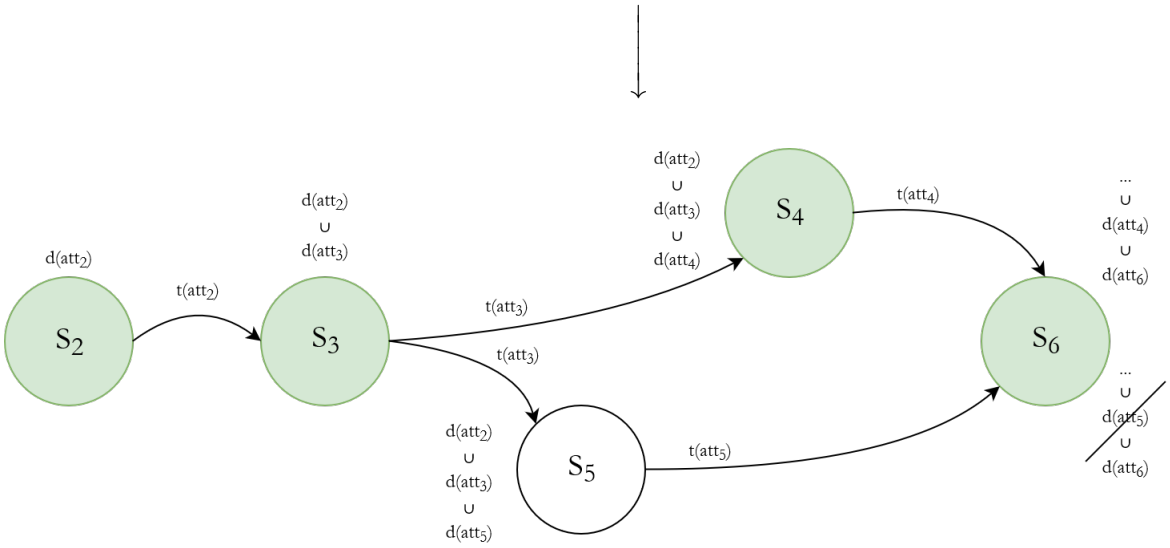


Fig. 4: Un percorso vincolati dai diritti richiesti

Il tempo del percorso è il tempo totale di attraversamento degli stati:

$$t(percorso) = t((S_2) \cup (S_3) \cup (S_4) \cup (S_6))$$

La probabilità del percorso è la somma delle probabilità dei singoli stati:

$$p(percorso) = p(S_2) + p(S_3) + p(S_4) + p(S_6) - (p(S_2) \cap p(S_3) \cap p(S_4) \cap p(S_6))$$

L'output dell'algoritmo sarà cosi rappresentato:

$$output = 1\ stato\ corrispondente, \{t(percorso), p(percorso)\}$$

Con una soluzione soltanto, tempo minimo e massimo coincidono.

1.3.2 Cercare eventuali stati futuri

L'ultimo requisito prevede di indicare tutti gli stati possibili a partire da uno dato.
Ipotizzando di essere arrivati a calcolare come da esempio precedente S_6, trovandoci su un grafo diretto aciclico, per scoprire i prossimi stati è sufficiente avviare un algoritmo *DepthFirstSearch (DFS)*.
Graficamente l'esplorazione è quella rappresentata sotto:

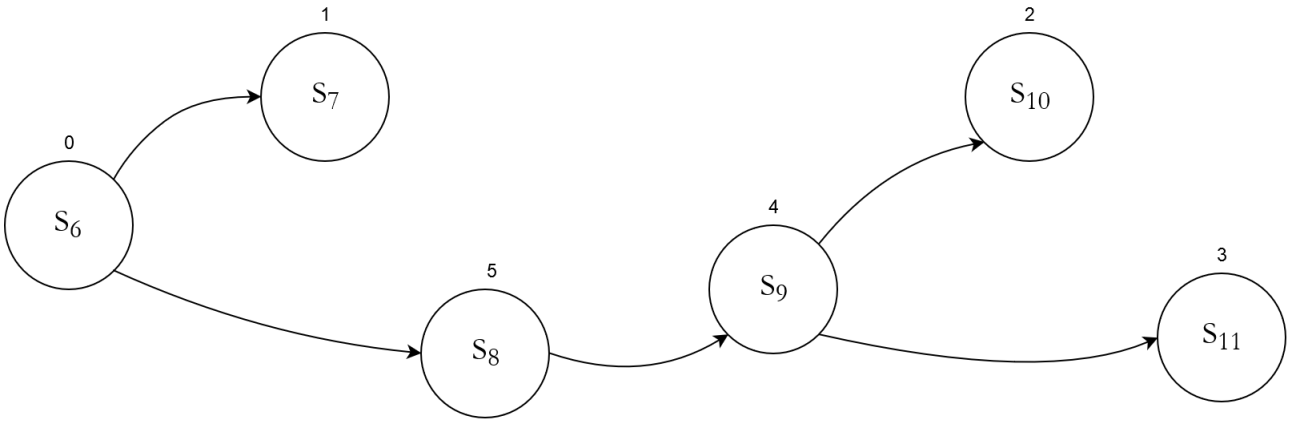


Fig. 5: Ricerca in profondità per scoprire altri stati

Viene restituita una lista di tutti gli stati raggiungibili da S_6:

$$DFS(S_6) = \{S_7, S_{10}, S_{11}, S_9, S_8\}$$

A questo punto se è necessario conoscere i dettagli come il tempo o la probabilità degli stati raggiungibili, è possibile applicare l'algoritmo $BreadthFirstTraversal$ descritto in **1.3.1**, per esempio per arrivare da S_6 a S_9:

$$Percorso = BreadthFirstTraversal(S_6, S_9) = \{S_6 \cup S_8 \cup S_9\}$$

2 Realizzazione

Di norma intrusion detection system é composto da 4 componenti.

- Dei **Sensori** che monitorano lo stato del sistema, notificando cambiamenti e alterazioni al *ids*.
- Un **Database** contenente una raccolta di attacchi conosciuti e attività maligna.
- Un **Algoritmo** in grado di elaborare i dati raccolti dai sensori e capace di utilizzare il database fornito per determinare se sia possibile passare da uno stato ad un altro, tra quelli conosciuti e catalogati nel database.
- L'**Interfaccia** che consente all'operatore di controllare l'attività del *ids*, visualizzarne i log e modificarne eventuali parametri.

Gli *ids* si dividono in due macrocategorie, quelli **passivi** osservano lo stato del sistema e informano l'operatore in caso di attività sospette. Gli *ids* **attivi** intervengono autonomamente in caso di rilevamenti positivi, ad esempio collaborando con un firewall. Questo programma é passivo e si limita a interpretare dati reali o quelli forniti dall'utente.

L'implementazione é un sistema multi-piattaforma consultabile sia in locale che da remoto tramite una WebApp.
Con un database accessibile dalla rete é possibile consultare da multiple istanze dell'algoritmo di valutazione lo stesso set di dati evitando al contempo problemi di aggiornamento per su singola macchina.
In particolare l'ecosistema *MySQL* per basi di dati relazionali permette di istanziare un database connesso alla rete in modo sicuro e facile, su tutte le piattaforme attuali.

Grazie all'ampia diffusione, *Python* e *PHP* vantano la presenza dei loro interpreti su gran parte dei calcolatori, rendendo conveniente con essi rispettivamente l'implementazione dell'algoritmo *ids* e dell'interfaccia utente.
Un ulteriore vantaggio é dato dalla presenza di connettori con database *MySQL* nelle librerie standard dei due linguaggi sopracitati.

In pratica un sistema è cosi composto:

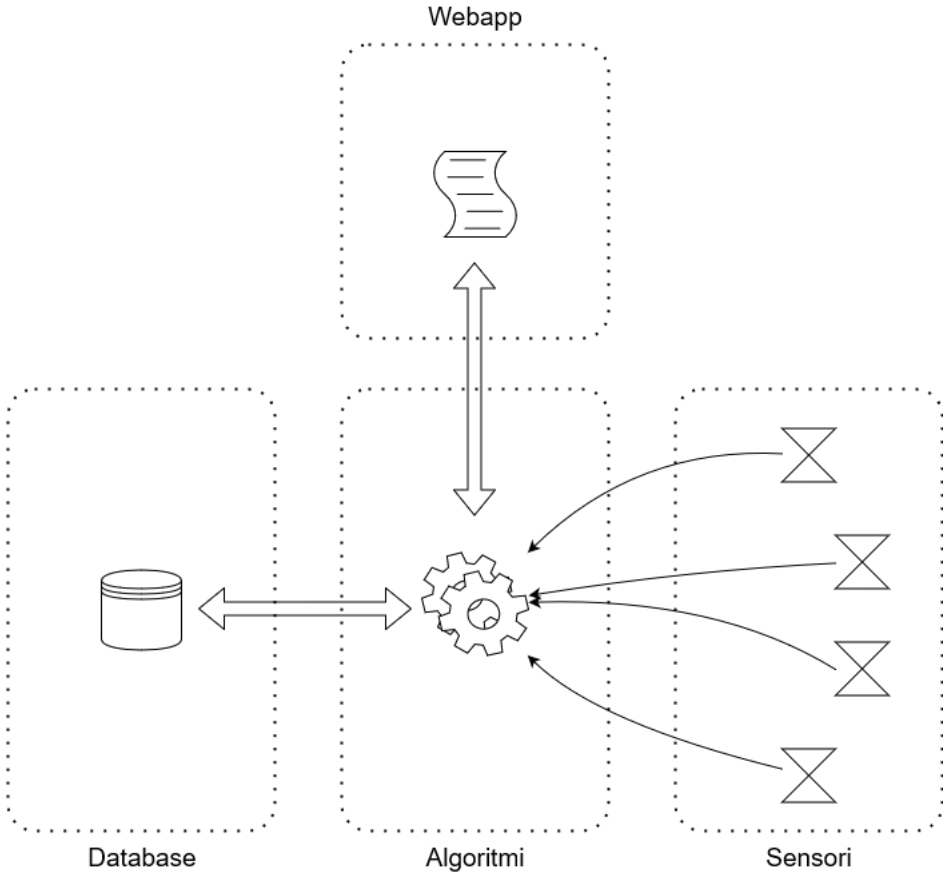
- Un server con database *SQL*.
- Un calcolatore che ospita l'interprete dell'algoritmo e il server *PHP* con una interfaccia per l'operatore.
- Un insieme di sensori che inoltrano al calcolatore sopra lo stato attuale del sistema.

In alternativa é possibile distribuire gli elementi in una rete, o istanziare l'algoritmo in ogni calcolatore, mantenendo il database, la gestione e l'interfaccia centralizzati.

2.1 Struttura

Il nostro sistema risiede interamente su un calcolatore, ma è distribuibile se necessario. In particolare è possibile disaccopiare il database e utilizzare le informazioni contenute in esso per istanze multiple.

L'interprete *PHP* in loco genera una interfaccia dinamica raggiungibile tramite l'indirizzo IP del calcolatore sul browser e si occupa di richiamare i singoli programmi contenenti l'algoritmo descritto in **1.3**. Questi programmi a loro volta comunicano con i sensori e il database.



I sensori generano inoltre dei log riguardanti la loro attività che vengono archiviati all'interno del database.
Lo storico dell'attività dei singoli sensori è consultabile dall'utente tramite interfaccia web. L'utente ha la possibilità di collegarsi da remoto, mentre i sensori devono essere attivati localmente.
In questa particolare implementazione i sensori si occupano di monitorare alcuni file selezionati dall'utente residenti sul medesimo file system dove é presente il programma.
Nel monitoraggio l'interesse é focalizzato sull'attributo dei permessi di cui gode ogni file residente in un filesystem di un sistema *UNIX* e *UNIX-Like*, il quale ci comunica di quali diritti godono determinati utenti rispetto al file stesso.

In un sistema Unix i permessi sono espressi in formato ottale, ovvero tre interi compresi tra 0 e 7[1].

Il sistema di riferimento é *Ubuntu 16.04 LTS* con server web *apache* e database *MariaDB*.

[1] NERSC - `https://docs.nersc.gov/filesystems/unix-file-permissions/`

2.2 Implementazione del database

Il set di dati é fornito sotto la seguente forma:

- ATTACCHI (*attacco*$_1$, *attacco*$_2$, *attacco*$_3$, ..., *attacco*$_N$, *probabilitá*)

Il funzionamento del programma é supportato dalla seguente base di dati, che include anche il funzionamento dei sensori (in formato *Entità-Relazione*):

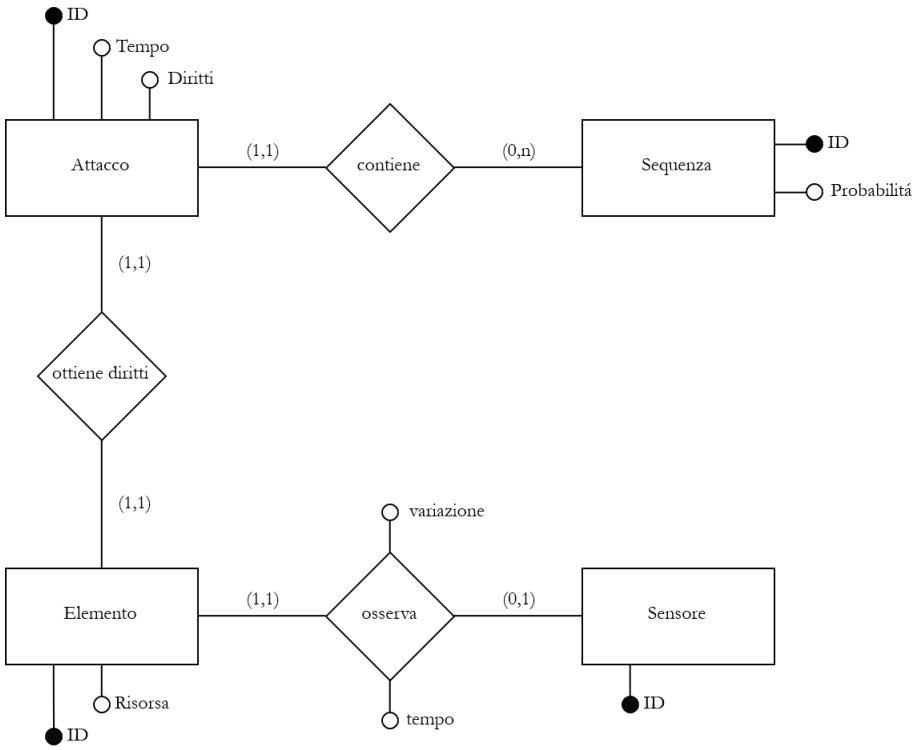


Fig. 6: Diagramma E-R della implementazione

2.2.1 Applicazione

Riducendo la lo schema E-R in terza forma normale:

- SEQUENZA (idSequenza,probabilitá)
- DESCRIZIONE-ATTACCO (idAttacco,sensore,diritti,tempo)

  chiave esterna: sensore → SENSORI(idSensore)
- ATTACCHI (attacco,idSequenza,idAttacco,posizione)

  chiave esterna: idSequenza → SEQUENZA(idSequenza)

  chiave esterna: idAttacco → DESCRIZIONE-ATTACCO(idAttacco)

La relativa implementazione in linguaggio SQL:

SEQUENZA (idSequenza,probabilitá)

```
CREATE TABLE ‘sequenza‘ (
  ‘idSequenza‘ INT NOT NULL AUTO_INCREMENT,
  ‘probabilita‘ INT NOT NULL,
  PRIMARY KEY (‘idSequenza‘);
```

DESCRIZIONE ATTACCO (idAttacco,sensore,diritti,tempo)

```
CREATE TABLE ‘descrizione_attacco‘ (
  ‘idAttacco‘ INT NOT NULL AUTO_INCREMENT,
  ‘sensore‘ VARCAR(45),
  ‘diritti‘ VARCAR(45),
  ‘tempo‘   VARCAR(45),
  PRIMARY KEY (‘idAttacco‘),
   FOREIGN KEY (‘sensore‘)
     REFERENCES ‘sensori‘ (‘idSensore‘)
     ON DELETE CASCADE
     ON UPDATE CASCADE;
```

ATTACCHI (attacco,idSequenza,idAttacco,posizione)

```
CREATE TABLE ‘attacchi‘ (
  ‘attacco‘ INT NOT NULL AUTO_INCREMENT,
  ‘idSequenza‘ INT NOT NULL,
  ‘idAttacco‘ INT NOT NULL,
  ‘posizione‘ INT NOT NULL,
  PRIMARY KEY (‘attacco,),
    FOREIGN KEY (‘idSequenza‘)
     REFERENCES ‘sequenza‘ (‘idsequenza‘)
     ON DELETE CASCADE
     ON UPDATE CASCADE,
    FOREIGN KEY (‘idAttacco‘)
     REFERENCES ‘descrizione_attacco‘ (‘idAttacco‘)
     ON DELETE CASCADE
     ON UPDATE CASCADE);
```

2.2.2 Sensori

- SENSORI (idSensore,elementoOsservato)
- DATI-SENSORI (idLettura,sensore,variazione,tempo)
  chiave esterna: sensore → SENSORI(idSensore)

In linguaggio SQL:

```
CREATE TABLE ‘sensori‘ (
  ‘idSensore‘ INT NOT NULL AUTO_INCREMENT,
  ‘elementoOsservato‘ VARCHAR(450) NOT NULL,
  PRIMARY KEY (‘idSensore‘);

CREATE TABLE ‘dati-sensori‘ (
  ‘idLettura‘ INT NOT NULL AUTO_INCREMENT,
  ‘sensore‘ INT NOT NULL,
  ‘variazione‘ VARCHAR(450) NOT NULL,
  ‘tempo‘ TIMESTAMP NOT NULL,
  PRIMARY KEY (‘idLettura,),
    FOREIGN KEY (‘sensore‘)
     REFERENCES ‘sensori‘ (‘idSensore‘);
```

Tutte le tabelle finora implementate operano essenzialmente in sola lettura e vengono attivate in scrittura soltanto all'aggiunta di nuovi attacchi nelle sequenze.
Eccezione fatta per la tabella *dati-sensori* la quale viene aggiornata ogni qual volta un sensore riporta una variazione. Le tuple aggiunte, nel caso si osservino file di sistema, potrebbero essere decine di migliaia al giorno. Per evitare una crescita esagerata può essere opportuno impostare la cancellazione delle tuple con timestamp vecchio, passato un certo lasso di tempo.

2.3 Implementazione delle query

La query restituisce una tabella con un attacco per riga, ad ogni attacco é associata la sequenza a cui appartiene insieme alla sua posizione in essa. Vengono dati gli attributi *tempo* e *probabilità* del singolo attacco.

$$\pi_{\text{idSequenza, posizione, idAttacco,diritti,tempo,probabilita}}((\text{sequenza\_attacchi}) \bowtie (attacchi) \bowtie (descrizione\_attacco)) \tag{1}$$

La relativa query in linguaggio SQL:

```
PROCEDURE ‘listaAttacchi‘()‘
   SELECT
       idsequenza,
       posizione,
       idattacco,
       diritti,
       tempo,
       probabilita
   FROM
    sequenza_attacchi SA
        INNER JOIN
    attacchi ATT on ATT.idSequenza = SA.idSequenza
        INNER JOIN
    descrizione_attacco DA on ATT.idAttacco = DA.idAttacco
```

2.4 Esecuzione del database

Viene restituita una tabella dal seguente aspetto:

```
mysql> call listaAttacchi();
+----------+-----------+---------+-----------------------+-------+-------------+
| Sequenza | posizione | Attacco | diritti               | tempo | probabilita |
+----------+-----------+---------+-----------------------+-------+-------------+
|        1 | 1         |       1 | /home/fileEsempio=644  | 0.05  | 0.17        |
|        1 | 2         |       3 | /home/fileEsempio1=764 | 0.13  | 0.17        |
|        1 | 3         |       6 | /home/fileEsempio2=644 | 0.55  | 0.17        |
|        1 | 4         |      11 | /home/fileEsempio2=644 | 0.44  | 0.17        |
|        1 | 5         |      19 | /home/fileEsempio3=740 | 0.30  | 0.17        |
|        2 | 1         |       6 | /home/fileEsempio4=644 | 0.55  | 0.21        |
+----------+-----------+---------+-----------------------+-------+-------------+
```

2.5 Importazione dei dati e realizzazione del grafo

L'importazione dei dati avviene tramite i seguenti passaggi:

1. Connessione al database
2. Ricezione dei dati
3. Conversione in strutture dati usabili da python
4. Costruzione del grafo

2.5.1 Preparazione dei dati

Il database SQL viene interrogato dal programma utilizzando la libreria *mysql.connector*.
La funzione `richiediDati(sql)` restituisce un oggetto contenente i dati restituiti dal server sotto forma di tabella.

```
def richiediDati(sql):
  mysqldb = mysql.connector.connect(user ='user',
                                    password= 'password',
                                    host = '127.0.0.1',
                                    port='3306',
                                    database='dbname')
  dbcursor = mysqldb.cursor(buffered=True)
  dbcursor.execute(sql)
  mysqldb.close()
  return dbcursor.fetchall()
```

Per poter immagazzinare i dati ricevuti vengono creati 4 dizionari e una classe apposita:

```
dizionarioAttacchi = dict()
listaTempi = dict()
listaProbabilita = dict()
listaDiritti = dict()
```

Tutti i dizionari condividono la medesima chiave, l'ID del attacco. Assumendo n come identificatore si un attacco si ha:

```
listaTempi[n]        # tempo di n
listaProbabilita[n] # probabilita di n
listaDiritti[n]      # diritti acquisiti in n
```

La classe vertice viene utilizzata per salvare i parametri salienti di un attacco e permette anche la successiva creazione del grafo e il suo parsing. Nel vettore `adiacenti` vengono raggruppati gli attacchi(vertici) successivi.

```
class Vertice:
  adiacenti = []
  Diritti = []
  def __init__(self,dir,prob,idx,att,seq,tmp = 1):
    self.Attacco = att
    self.Diritti = dir
    self.Probabilita = prob
    self.Tempo = tmp
    self.Indice = idx
    self.Sequenza = seq
  def sequenza(self):
    return self.Sequenza
```

La seguente funzione, `elaboraDatiTabella(querySQL)`, genera un array di sequenze.

```
def elaboraDati(tabella):
  Out = []
  for row in richiediTabella(tabella):
    vertice = Vertice(row[3],float(row[5]),int(row[1]),int(row[2]),int(row[0]),float(
                                            row[4]))
    listaTempi[row[2]] = row[4]
    listaDiritti[row[2]] = row[3]
    dizionarioStati[row[3]] = row[2]

    if row[2] not in listaProbabilita:
      listaProbabilita[row[2]] = float(row[5])
    else:
      tmp = listaProbabilita[row[2]]
      listaProbabilita[row[2]] = float(row[5]) + tmp - (float(row[5])*tmp)
    Out.append(vertice)

  sequenze = dict()

  for vertice in Out:
    if vertice.Sequenza not in sequenze:
      sequenze[vertice.Sequenza] = [vertice]
    else:
      sequenze[vertice.Sequenza].append(vertice)
  return sequenze
```

Il selettore `row[n]` permette la navigazione nella tupla appena estratta dal database tramite la funzione `richiediTabella`. L'aspetto dei dati appena importati è il seguente come nella query **2.3**:

$$[idsequenza, posizione, idattacco, diritti, tempo, probabilita]$$

La prima parte della funzione restituisce un vettore non ordinato in cui ogni elemento è un attacco. Nella seconda parte viene generato un dizionario in cui la chiave è l'id delle sequenze. Vengono cosi ricostruite le sequenze di attacchi originali.
La complessitá delle funzioni in **2.5.1** é lineare rispetto alla quantitá di dati.

2.5.2 Costruzione del grafo

Per la costruzione del grafo vero e proprio, la seguente funzione scorre le singole sequenze e genera gli archi verificando la consequenzialità degli attacchi come visto in **1.3** in ogni sequenza. Viene creato un dizionario (hash table) che ha per chiave lo stato e come valore una lista degli stati successivi ad esso.

1. Seleziono una sequenza
2. Leggo uno `stato[n]`
3. Verifico se `stato[n]` é gia presente nel dizionario
4. Se si, aggiungo ad esso `stato[n+1]` e passo al punto 6
5. Altrimenti creo una nuova chiave nel dizionario di `stato[n]` e aggiungo ad esso `stato[n+1]`
6. Se si tratta del penultimo stato vado al punto 8
7. Riparto dal punto 2 con `stato[n+1]`
8. Esamino una nuova sequenza e torno al punto 1

```
def costruisciGrafo(S):
  grafo = dict()
  for sequenze in range(len(S)):
    for stato in range(len(S[sequenze])-1):
      if(S[sequenze][stato][3] in grafo):
        if(S[sequenze][stato+1][3] not in grafo[S[sequenze][stato][3]]):
          grafo[S[sequenze][stato][3]].append(S[sequenze][stato+1][3])
      else:
        grafo[S[sequenze][stato][3]] = []
        grafo[S[sequenze][stato][3]].append(S[sequenze][stato+1][3])
  for vertice in grafo:
    grafo[vertice] = sorted(grafo[vertice])
  return grafo
```

Un ipotetico output potrebbe essere il seguente:

```
sequenze = {(1: 1, 2, 3, 5, 7),
            (2: 1, 4, 7, 9, 10),
            (3: 3, 6, 9, 11)}

grafo = costruisciGrafo(sequenze)

print(grafo)
>{1:[2,4],2:[3],3:[5,6],4:[7],5:[7],6:[9],7:[9],9:[10,11]}
```

La rappresentazione è quella standard:

$$vertice : [adiacenti]$$

2.6 Algoritmo

L'algoritmo precedentemente definito in **1.3** come *BreadthFirstTraversalConVincolo* viene implementato come unione di due funzioni.

2.6.1 Breadth First Traversal

BreadthFirstTraversal restituisce una lista di possibili percorsi dal primo all'ultimo attacco dello stato cercato.

Il Breadth First Traversal é un algoritmo ricorsivo e deriva dal Breath First Search (BFS) ovvero *ricerca in ampiezza*.
La complessità dell'algoritmo é $O(Vertici + Archi)$, la somma dei vertici e degli archi.

Nella chiamata si specifica il vertice d'inizio e quello finale. I passi logici sono i seguenti:

1. Aggiungo il vertice corrente al percorso
2. Se l'inizio coincide con la fine restituisco il percorso
3. Se l'inizio non é presente nel grafo restituisco un percorso vuoto
4. Genero una lista di nuovi percorsi per ogni nodo adiacente
5. Per ogni vertice adiacente richiamo il punto 1
6. Aggiungo i nuovi percorsi appena trovati alle liste generate al punto 4
7. Restituisco la lista dei percorsi

```
def breadthFirstTraversal(grafo, inizio, fine, percorso=[]):
        percorso = percorso + [inizio]
        if inizio == fine:
            return [percorso]
        if inizio not in grafo:
            return []
        percorsi = []
        for vertice in grafo[inizio]:
            if vertice not in percorso:
                nuoviPercorsi = trovaPercorsi(grafo, vertice, fine, percorso)
                for nuovoPercorso in nuoviPercorsi:
                    percorsi.append(nuovoPercorso)
        return percorsi
```

Di seguito un esempio di output:

```
Vertici = {2,3,4,5,6}
Archi = {2->3,3->4,4->6,3->5,5->6}
Grafo = (Vertici,Archi)
Input = {(2,4,6)}

print breadthFirstTraversal(Grafo,inizio(Input),fine(Input))
> {(2,3,4,6),(2,3,5,6)}
```

2.6.2 Applicazione dei vincoli

La seconda funzione filtra la lista dei possibili percorsi restituiti dalla funzione sopra (*BreadthFirstTraversal*) applicando i restanti vincoli dello stato in input come da esempio visto in **1.3**, scartando dalla lista i percorsi non compatibili.

`lista[:n]` Rappresenta la porzione inferiore della lista rispetto all'indice `n` mentre `lista[n:]` quella superiore.
`Input` é la lista contenente i vertici dello stato da cercare.

1. Seleziono un possibile percorso
2. Verifico che in `percorso[:n]` sia presente `Input[n]`
3. Se si, passo a `Input[n+1]` verificando che sia presente in `percorso[n:]` e cosi via
4. Se riesco ad esaurire tutti i vertici di `Input` restituisco il percorso altrimenti riparto da 1

La complessità dell'algoritmo é al più $O(Vertici)$, in ogni caso minore o uguale alla funzione **2.5.1**.

I passaggi compiuti dalla funzione sono i seguenti:

```
def applicaVincoli(proposte,richiesta):
  percorsiValidi = dict()
  for proposta in proposte:
    richiestaTMP = []
    richiestaTMP.extend(richiesta)
    for vertice in proposta:
      if vertice in richiestaTMP:
        richiestaTMP.remove(vertice)
      if not richiestaTMP:
        percorsiValidi[str(proposta)] = proposta

  return list(percorsiValidi.values())
```

Completando l'esempio di prima (**2.5.1**):

```
Vertici = {2,3,4,5,6}
Archi = {2->3,3->4,4->6,3->5,5->6}
Grafo = (Vertici,Archi)
Input = {(2,4,6)}

print breadthFirstTraversal(Grafo,inizio(Input),fine(Input))
> {(2,3,4,6),(2,3,5,6)}

print applicaVincoli(breadthFirstTraversal(Grafo,inizio(Input),fine(Input)),Input)
> {(2,3,4,6)}
```

2.6.3 Creazione del risultato

Ottenuti i percorsi validi all'interno del grafo, si procede a calcolarne i tempi e le probabiltà utilizzando il dizionario con gli attributi generato in **2.4.1** secondo le regole definite in **1.3**, ovvero:

$$t(percorso) = \ t(percorso) = \ t((S_2) \cup (S_3) \cup (S_4) \cup (S_6))$$

$$p(percorso) = \ p(S_2) + p(S_3) + p(S_4) + p(S_6) - (p(S_2) \cap p(S_3) \cap p(S_4) \cap p(S_6))$$

La funzione formatta l'output cosi:

`percorso: [tempo,probabilita,vertici]`

```
def calcolaTempi(percorsi,tempi,listaProbabilita):
  percorsiETempi = []
  for percorso in percorsi:
    tempo = 0.0
    probabilita = 0.0
    for nodo in percorso:
      tempo = tempo + float(tempi[nodo])
      probabilita = probabilita + listaProbabilita[nodo] - (probabilita*
                                          listaProbabilita[nodo])
    percorso.insert(0,probabilita)
    percorso.insert(0,str(tempo))
    percorsiETempi.append(percorso)
  return percorsiETempi
```

Un esempio di output:

```
percorsi = applicaVincoli(trovaPercorsi(Grafo,inizio(Input),fine(Input)),Input)

print(percorsi)
>{(1, 2, 3, 4, 5, 6, 7, 12), (1, 2, 3, 4, 6, 7, 12)}

print(calcolaTempi(percorsi,attributi))
>{1:[31.4,'p0.99', 1, 2, 3, 4, 5, 6, 7, 12], 2:[30.9,'p0.98', 1, 2, 3, 4, 6, 7, 12]}
```

A questo punto per ricavare il numero di soluzioni e le soluzioni minime e massime é sufficiente utilizzare le funzioni standard di python `max()`, `min()` e `len()`.

```
  print("percorsi totali:",str(len(percorsi)))
  for value in percorsi.items():
    if(max(percorsi[0]) == value[0]):
      print("percorso massimo: ",value[2:]," secondi: ", value[0], "prob:",value[1])
    if(min(percorsi[0]) == value[0]):
      print("percorso minimo: ",value[2:]," secondi: ", value[0],"prob:",value[1])
```

Avente questo output:

```
percorsi totali:2
percorso massimo:  [1, 2, 3, 4, 5, 6, 7, 12]  secondi:  31.4 prob: 0.99
percorso minimo:  [1, 2, 3, 4, 6, 7, 12]  secondi:  30.9 prob: 0.98
```

2.6.4 Generare l'output

Formattando opportunamente lo *stdout* si permette all'interprete php di includere l'output del programma in un documento *HTML*. Per visualizzare correttamente il risultato prodotto in **2.6.3** si procede nel seguente modo:

- Stampo l'intestazioe e il numero di soluzioni trovate
- Dichiaro una tabella `<table>`
- Per ogni percorso dichiaro una riga `<tr></tr>`
- Stampo il tempo (`percorso[0]`) e la probabilità (`percorso[1]`)
- Per ogni percorso evidenzio tutti i diritti acquisiti e il loro tempo (`for attacco in percorso..`)
- chiudo la tabella `</table>`

```
print("<h3>Stati</h3>")
print("<h5>Soluzioni trovate: " + str(soluzioni) + "</h5>")
print("<table class=\"table table-striped ...\"><tr><th>...</th></tr>")

for percorso in percorsi:
  print("<tr><td>")
  print(str(round(float(percorso[0]),4)))    #estraggo il tempo alla posizione 0
  print("</td><td>")
  print(str(round(percorso[1],4)))           #estraggo la probabilita in [1]
  print("</td><td class=\"monos\">")
  t=0.0
  for attacco in percorso[2:]:               #scansiono il resto del vettore
    t = float(listaTempi[attacco]) + t
    print( "t=" + str(round(t,4)) + "  ->   " +  listaDiritti[attacco])
    print("<br>")
  print("</td></tr>")
print("</table>")
```

Il risultato interpretato dal browser si presenta cosi:

Stati

Soluzioni trovate: 4

tempo	probabilita'	diritti acquisiti
1.6	0.9021	t=0.05 -> /home/administrator/Desktop/test/aaa.txt=644 t=0.18 -> /home/administrator/Desktop/test/aaa.txt=764 t=0.73 -> /home/administrator/Desktop/test/b=644 t=1.17 -> /home/administrator/Desktop/test/CCCCC=644 t=1.47 -> /home/administrator/Desktop/test/G.tzt=740 t=1.6 -> /home/administrator/Desktop/test/zzzzZZzzz=667
5.02	0.8974	t=0.05 -> /home/administrator/Desktop/test/aaa.txt=644 t=0.18 -> /home/administrator/Desktop/test/aaa.txt=764 t=0.73 -> /home/administrator/Desktop/test/b=644 t=1.4 -> /home/administrator/Desktop/test/b=664 t=2.14 -> /home/administrator/Desktop/test/b=775 t=2.7 -> /home/administrator/Desktop/test/CCCCC=664 t=3.04 -> /home/administrator/Desktop/test/DDDDDDDD=775 t=3.58 -> /home/administrator/Desktop/test/G.tzt=775 t=4.39 -> /home/administrator/Desktop/test/gg=664 t=4.89 -> /home/administrator/Desktop/test/KKK=664 t=5.02 -> /home/administrator/Desktop/test/zzzzZZzzz=667

Fig. 7: Tabella con output

2.6.5 Ulteriori diritti raggiungibili

A partire dall'ultimo stato trovato con **2.6.1** ovvero *BFT*, possiamo istanziare una ricerca DFS per trovare tutti gli altri stati raggiungibili da esso. La funzione restituisce un *set*, una lista con in cui ogni elemento é unico, con tutti gli stati trovati. Viene restituita una lista con i nodi che si trovano successivamente al nodo `inizio` in un grafo aciclico diretto:

```
def DFS(grafo, inizio, visitati=None):
  if inizio not in grafo:
    visitati.add(start)
    return
  if visitati is None:
    visitati = set()
  visitati.add(inizio)
  for adiacente in grafo[inizio]:
    if adiacente not in visitati:
      dfs(grafo, adiacente, visitati)
  return visitati
```

Per generare l'output contenente i diritti eventualmente raggiungibili, si controlla innanzitutto se ci sono dei *Stati* su cui lavorare (`percorsi`), altrimenti si annulla l'operazione (`return`).

```
  if not stati:
    return
  else:
    dirittiRaggiungibili = dfs(G,percorsi[0][-1])
    dirittiRaggiungibili.remove(percorsi[0][-1])     #rimuovo il vertice di partenza
```

In caso positivo, procedo a generare uno *stdout* HTML conforme per ottenere un risultato analogo a **2.6.4**:

```
  print("<h3>Diritti raggiungibili</h3>")
  print("<h5>Soluzioni trovate: " + str(len(dirittiRaggiungibili)) + "</h5>")
  print("<table class=\"table table-striped ...\"><tr>...</tr>")

  #per ogni nodo trovato da DFS..
  for diritto in dirittiRaggiungibili:

    #trovo la distanza e il percorso a partire dallo stato di inizio
    percorso = trovaPercorsi(G,stati[0][-1],item)

    #calcolo probabilita e tempi, come in 2.6.4
    percorsoValutato = calcolaTempi(percorso,listaTempi,listaProbabilita)

    #estraggo soltato il piu lento e il piu veloce
    percorsoValutato = trovaMinimoEMassimo(percorsoValutato)

    #compongo cosi la riga della tabella: tempo, probabilita, diritti
    print("<tr><td>" + str(round(float(percorsoValutato[0][0]),4)) + "</td>")
    print("<td>" + str(round(percorsoValutato[0][1],4)) + "</td>")
    print("<td class=\"monos\">" + listaDiritti[diritto] + "</td></tr>")

  print("</table>")
```

2.7 Sensori

Il controllo dei singoli file é delegato alla utility open source *inotifytools* che crea una interfaccia con il kernel e capta eventi come variazioni di diritti o modifiche ai file.

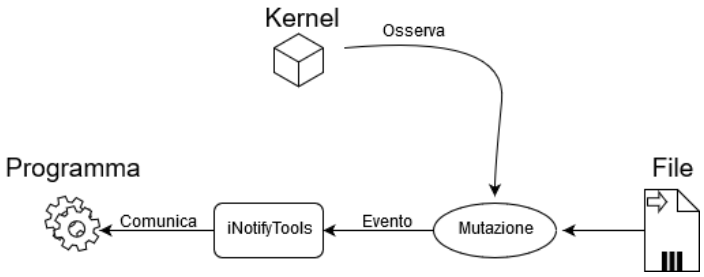


Fig. 8: Schema del funzionamento di un sensore

2.7.1 Catturare gli eventi

Per iniziare ad osservare un file é sufficiente avviare lo script *monitor.sh* fornendo come argomento il file con il relativo percorso:

./monitor.sh /home/fileEsempio.txt

Man mano che capta gli eventi, li inoltra al programma *monitor.py* che a sua volta si occupa di generare il log e inoltrare il tutto verso il programma principale. Lo script é cosi composto:

cat monitor.sh

```
inotifywait -mq  $1 |
while IFS= read -r events; do
  python3 /var/www/html/sensori/monitor.py $events
done
```

Il programma *monitor.py* essenzialmente si occupa di inserire la stringa dell'argomento nel database per creare il log.

```
def inviaDati(sql): #crea la connessione con il database
  mydb = mysql.connector.connect(user ='root', password= 'password', host = '127.0.0.
                                        1',port='3306', database='app')
  dbcursor = mydb.cursor(buffered=True)
  dbcursor.execute(sql)
  mydb.commit()
  mydb.close()

def main(sys_array):
  #decodifica i diritti del file in formato ottale ed esegue la query
  diritti = oct(os.stat(sys_array[1])[ST_MODE])[-3:]
  sql = "insert into dati_sensori(sensore,variazione,diritti) values(" + "\"" +
                                        sys_array[1] + "\",\"" + sys_array[2] + "
                                        \",\"" + str(diritti) + "\")"
  inviaDati(sql)
  return

main(sys.argv)
```

2.7.2 Leggere lo stato del sistema

Lo script `stato_attuale.py` richiede al database la lista dei file osservati e interroga attraverso la libreria `stat` i diritti dei singoli file al momento della richiesta restituendo una tabella formattata in html attraverso lo stdout.

```
import sys
import mysql.connector
import stat
import os

def main():
  stato = []
  sql = "select elemento_osservato from sensori"

  for row in richiediTabella(sql):
    diritti = [int,str]
    diritti[0] = row[0]
    diritti[1] = str(oct(os.stat(row[0])[ST_MODE])[-3:])
    stato.append(diritti)

  print("<table class=\"table ..."><tr><td>elemento</td><td>diritti</td></tr>")
  for i in stato:
    #elemento osservato = i[0]
    #i[1] = diritti di i[0]
    print("<tr class=\"monos\"><td>" + i[0] + "</td><td>" + i[1] + "</td></tr>")
  print("</table>")

main()
```

In particolare la libreria `stat`[2], presente anche in **2.7.1**, viene utilizzata per interrogare gli *inode* alla base del file system UNIX.

[2]Documentazione Python - https://docs.python.org/3.0/library/stat.html

2.8 Tempi

I rilevamenti dei tempi di esecuzione[3] degli script sono stati realizzati tramite il comando *time*[4]. Il database contiene circa 300 attacchi suddivisi in 12 sequenze.

```
>time python3 ricerca_simulatore.py
real 0m0.057s
user 0m0.038s
sys     0m0.015s

>time python3 stato_attuale.py
real 0m0.051s
user 0m0.046s
sys   0m0.004s
```

Per una visione più specifica, utilizzando la libreria python `time`[5], é possibile monitorare il tempo impiegato dalle singole funzioni nei programmi. In particolare può essere letto il tempo attuale dal contatore di sistema tramite il metodo `time.time()`, poi attraverso riletture e le relative sottrazioni vengono calcolati i tempi delle singole funzioni. Una variante appositamente modificata del file `ricerca_simulatore.py` riporta i seguenti valori:

```
>python3 ricerca_simulatore_tempi.py
tempo totale:
0.008s
tempo query:
0.0056s
tempo costruzione grafo:
0.0001s
tempo ricerca percorsi:
0.0018s
tempo generazione output:
0.0005s
```

Come é possibile osservare, la ricezione dei dati dal database occupa quasi interamente il tempo di esecuzione. Per ottimizzare questo aspetto, visto il cambiamento sporadico dei dati, é possibile generare una cache locale che viene aggiornata durante l'aggiunta o rimozione di dati, per esempio tramite l'utilizzo della libreria `pickle`[6].

[3] Ubuntu 16.04 in ambiente virtuale VMware su unità SSD, processore i5-4310U
[4] Programma Unix - vedi "man time" da bash
[5] Documentazione Python - https://docs.python.org/3/library/time.html
[6] Documentazione Python - https://docs.python.org/3/library/pickle.html

2.8.1 Variante con cache locale

La libreria `pickle` permette di salvare su una memoria di massa locale una struttura dati qualsiasi generata in un programma python.

Si evita cosi di appesantire il flusso del programma con la connessione al database remoto e la relativa richiesta di dati, effettuando quest'ultimo passaggio solamente in caso di aggiunta o cancellazione di dati dalle tabelle.
Al programma viene cosi permesso di leggere la struttura già elaborata direttamente dalla memoria locale che lavora nell'ordine di pochissimi millisecondi.

Il programma `ricerca_simulatore` viene diviso in due sotttoprogrammi, uno che si occupa di aggiornare la cache quando necessario e l'altro atto ad eseguire l'algoritmo di ricerca sul grafo letto dalla memoria locale.

```
>time python3 ricerca_simulatore_cache.py
real 0m0.053s
user 0m0.037s
sys     0m0.011s
```

Sui risultati, ottenuti nel medesimo sistema di **2.8**, é possibile osservare un miglioramento nel tempo di esecuzione. In particolare il tempo occupato nella sezione `sys` ha subito un netto miglioramento, operazioni come l'accesso al disco e alle interfacce di rete non sono possibili in modalità *user*[7].

2.8.2 Tempi in relazione alla dimensione del database

Per avere una stima delle prestazioni del programma in ambiti realistici é stata effettuata una nuova misurazione incrementando il numero di attacchi da 300 a circa 5000. Di seguito i risultati con e senza l'utilizzo della cache.

```
>time python3 ricerca_simulatore.py        #con accesso diretto al DB
real 0m0.162s
user 0m0.137s
sys     0m0.042s

>time python3 ricerca_simulatore_cache.py  #con cache
real 0m0.150s
user 0m0.122s
sys     0m0.029s
```

[7]UNIBO. Micaela Spigarolo - `http://www.cs.unibo.it/~montreso/so/lucidi/lso-05-uml-4p.pdf`

3 WebAPP

La libreria base utilizzata per disegnare l'interfaccia utente é *Bootstrap*[8] che permette di realizzare un sito web dall'aspetto omogeneo e adattivo. La raccolta contiene codice e modelli di progettazione CSS facilmente richiamabili tramite classi, mantenendo al contempo il codice HTML pulito e sgravando il programmatore dal lavoro sui fogli di stile.
Bootstrap organizza il layout della pagina in contenitori.

3.1 Struttura della pagina

Per garantire una omogeneità nella generazione dell'interfaccia utente ed evitare ripetizioni nel codice, tutte le pagine sono composte in gran parte da codice comune ricomposto tramite *php* lato server. L'unica componente variabile di una pagina é il contenuto di `<div class="main-content">`.

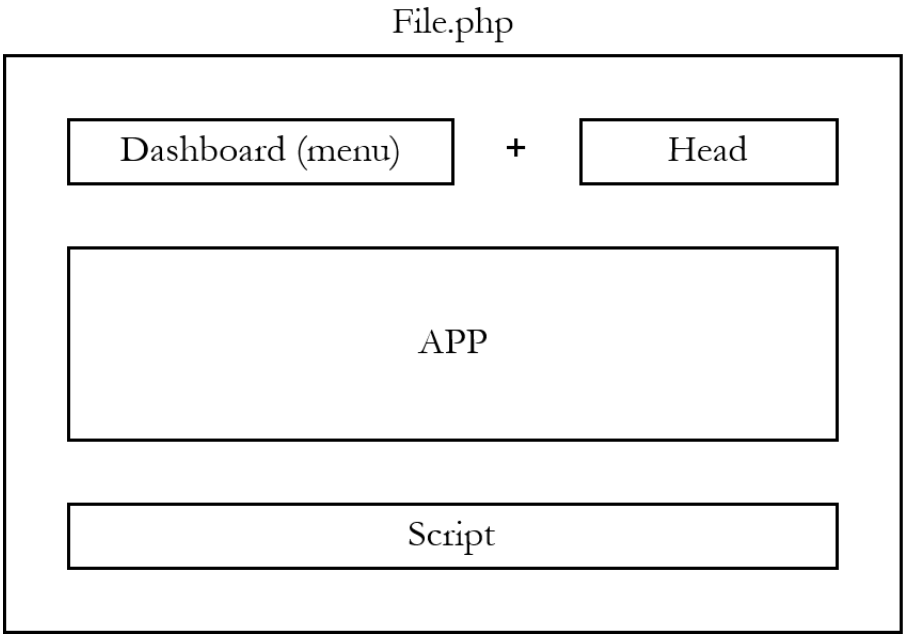


Fig. 9: Schema della struttura di un file *php* del programma

Ovvero un file minimo é cosi strutturato:

```
<?php include('common_head.php'); ?>
<?php include('common_dash.php'); ?>
  <div class="main-content">
   <p>
    contenuto
   </p>
  </div>
<?php include('common_end.php'); ?>
```

[8] Sito ufficiale Bootstrap - https://getbootstrap.com/

3.1.1 Head

I contenuti del head rimangono soltanto quelli essenziali, esclusi gli script, i quali per questioni di performance é preferibile caricare a fine pagina[9]. Il codice si occupa oltre all'intestazione, anche dell'apertura della tag `<body>`.

```
//file: common_head.php

<!DOCTYPE html>
<html lang="it">
<head>
  <link href="css/style.css" media="all" rel="Stylesheet" type="text/css"/>
  <link href="css/bootstrap.css" media="all" rel="Stylesheet" type="text/css"/>
  <link href="css/jquery-ui.css" media="all" rel="Stylesheet" type="text/css"/>

  <title>applicazione</title>

  <link rel="shortcut icon" href="img/favicon.ico" type="image/x-icon">
  <link rel="icon" href="img/favicon.ico" type="image/x-icon">
</head>
<body>
```

3.1.2 Script

Il file **common.end** contiene i riferimenti alle librerie JavaScript e le tag di chiusura del documento.

```
//file: common_end.php

<script src="script.js"></script>
<script src="lib/jquery-min.js"></script>
<script src="lib/jquery-ui.js"></script>
</body>
</html>
```

[9]Risorse per sviluppatori Yahoo - https://developer.yahoo.com/performance/rules.html

3.1.3 Menu e interfaccia

La componente `<nav>` dell'interfaccia contiene l'intero menu e la barra superiore di navigazione. L'implementazione é quella standard di Bootstrap per pannelli amministrativi. Il primo corpo `<nav>` dichiara la barra superiore utilizzando la classe `navbar-fixed-top`. Il secondo crea il menu laterale tramite la classe `navbar-primary-menu`. Di seguito la struttura del file `common_dash.php` contenente il menu:

```
<nav class="navbar navbar-inverse navbar-global navbar-fixed-top">
   <div class="container-fluid">
     <div class="navbar-header">
       <span class="icon-bar"></span>
       <a class="navbar-brand" href="app.php">applicazione</a>
      </div>
       <div id="navbar" class="collapse navbar-collapse">
        <ul class="nav navbar-nav navbar-user navbar-right">
         <li><a href="app_impostazioni.php?sett=serv">Impostazioni</a></li>
        </ul>
      </div>
   </div>
</nav>
<nav class="navbar-primary">
  <ul class="navbar-primary-menu">
    <li>
      <a href="app.php"><span class="nav-label">Applicazione</span></a>
      <a href="app_simulazione.php"><span class="nav-label">Simulatore</span></a>
      <a href="app_sensori.php"><span class="nav-label">Sensori</span></a>
      <a href="app_dati.php"><span class="nav-label">Consulta dati</span></a>
      <a href="app_impostazioni.php"><span class="nav-label">Imp.</span></a>
    </li>
  </ul>
</nav>
```

Schematizzando il tutto:

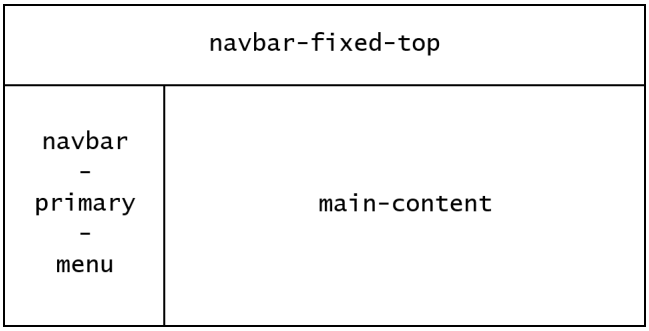


Fig. 10: Mockup della struttura visiva dell'interfaccia

3.2 Composizione

Il programma si divide in 4 sezioni, tutti i file principali portano il prefisso `app`. Le parti comuni portano il prefisso `common`. I restanti script *php* e i programmi python si trovano all'interno della cartella `backend`.

- Applicazione (`app.php`)

  Permette di visualizzare la lettura corrente dei sensori trova lo stato corrispondente.

  Viene chiamato lo script `get_dati_sensori.php` che chiama a sua volta `stato_attuale.py`

  Successivamente i dati acquisiti vengono elaborati dal programma `ricerca.py`

- Simulatore (`app_simulatore.php`)

  Permette di comporre uno stato

  Ne visualizza i tempi, la probabilità e la possibile evoluzione futura.

  Viene chiamato lo script `get_dati_simulatore.php` che chiama a sua volta `simulatore.py`

- Sensori (`app_sensori.php`)

  Permette di visualizzare la lista dei sensori e le relative letture insieme allo storico.

  Viene chiamato lo script `get_lista_sensori.php`

  Lo storico viene generato dallo script `get_storia_sensore.php`

- Dati (`app_dati.php`)

  Consente all'utente di navigare tra i dati riguardanti gli attacchi contenuti nel database

L'anteprima del programma completo:

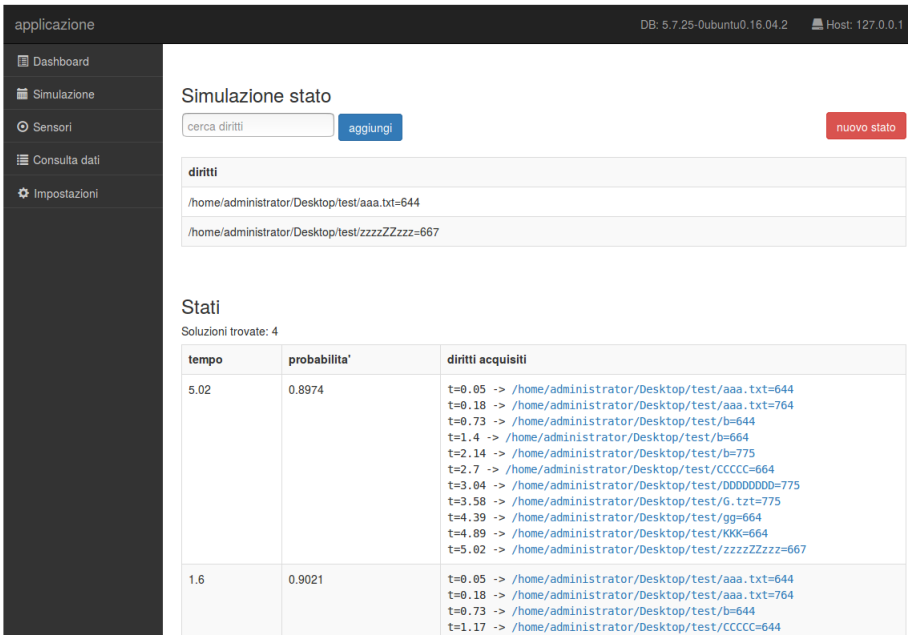


Fig. 11: Anteprima della pagina principale

3.3 Backend

3.3.1 Richieste asincrone Ajax

Tutte le richieste di dati asincrone sono basate sulla seguente funzione JavaScript utilizzando il metodo standard `XMLHttpRequest()`. Il parametro *s*, una stringa, é il valore "get" passato allo script *php* richiamato. Una volta ricevuta la risposta dal server, é possibile far assegnare i dati ricevuti ad un elemento del *DOM html*, per esempio utilizzando il metodo `innerHTML`. Tutte le funzioni JavaScript sono contenute nel file `scripts.js`.

```
function richiestaAjax(stringa) {
   if (stringa == "") {
       return;
   } else {
       xmlhttp = new XMLHttpRequest();
       xmlhttp.onreadystatechange = function() {
           if (this.readyState == 4 && this.status == 200) {
               //codice da eseguire al completamento della richiesta
               document.getElementById("esempio").innerHTML = this.responseText;
           }
       };
       xmlhttp.open("GET","backend/get_data.php?param=" + stringa,true);
       xmlhttp.send();
     }
}
```

3.3.2 Connessione al database

All'interno del file `backend/db_conn.php` vi sono le informazioni necessarie per stabilire una connessione con il database, il driver utilizzato é quello standard di *php*, `mysqli`. Il file deve essere incluso in ogni script che si interfaccia con il database.

```
<?php
$utente = "root";
$password = "password";
$server = "127.0.0.1";
$database = "app";

$conn = mysqli_connect($server,$utente,$password,$database);

if(!$conn) die("errore di connessione".mysqli_error($conn));
?>
```

3.3.3 Richiamare gli algoritmi

L'idea é di richiedere informazioni alla macchina che esegue il software tramite richieste asincrone come descritte in **3.2.1** ed integrare le risposte tramite il metodo `innerHTML` nella pagina del browser ad ogni richiesta o cambiamento dell'input dell'utente.

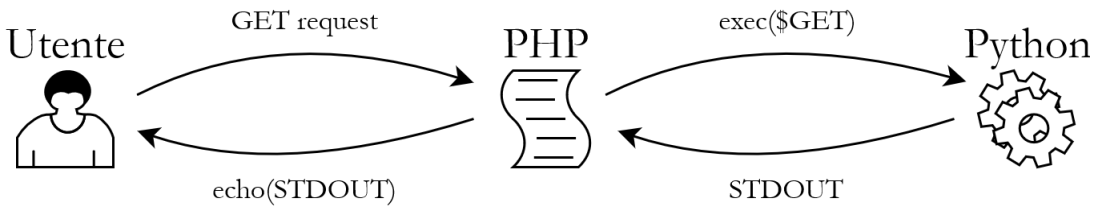


Fig. 12: Schema di una richiesta per richiamare un programma python dal web

Il programma prende come parametro la lista dei diritti dello stato, inviati dall'utente tramite il parametro *GET* di php. Da notare, il programma viene eseguito in una shell che appartiene probabilmente al gruppo `www-data` se si esegue un server *apache* e di conseguenza non gode di diritti di scrittura su disco per motivi di sicurezza.

```
<?php
//preparazione del comando
$parametri = $_GET['parametri'];

$script = 'python/script.py';
$python = 'python3';

//assegnazione della stringa "python3 python/script.py"
$comando = "$python $script $parametri";

//esecuzione dello script in una shell locale
//lo stdout viene salvato in $output
exec("$comando", $output);

//adattamento dello stdout come semplice stringa
echo implode($output);

//chiusura script php
die();
?>
```

Lo *standard output* del programma python é formattato utilizzando le opportune tag $HTML$ per consentire al browser, una volta ottenuta la stringa, di visualizzare i dati in maniera gradevole per l'utente come si può notare per esempio in **3.2.4** dove viene costruita una tabella semplicemente utilizzando il comando `print()`.

3.3.4 Interrogare lo stato

Il seguente programma controlla nel database quali sono gli elementi osservati dai sensori. Tramite la libreria **stat** è possibile più interrogare gli *inode* dei singoli file osservati e ricavare i diritti correnti in formato ottale (`oct(os.stat(row[0]))`). Viene poi generato uno stdout interpretabile da browser come in **2.6.4**:

```
import sys
import mysql.connector
import os
import stat

def richiediTabella(sql):
  mydb = mysql.connector.connect(user ='root', password= 'radeon', host = '127.0.0.1'
                                          ,port='3306', database='app')
  dbcursor = mydb.cursor(buffered=True)
  dbcursor.execute(sql)
  mydb.close()
  return dbcursor.fetchall()

def main():
  stato = []
  sql = "select elementoOsservato from sensori"
  for row in richiediTabella(sql):
    diritti = [int,str]
    diritti[0] = row[0]
    diritti[1] = str(oct(os.stat(row[0])[ST_MODE])[-3:])
    stato.append(diritti)

  print("<table class=\"table table-striped...\"><tr>...</tr>")
  for i in stato:
    print("<tr class=\"monos\"><td> + i[0] + "</td><td>" + i[1] + "</td></tr>")
  print("</table>")

main()
```

Visualizzata nel browser la tabella si presenta cosi:

Stato attuale

sensore	diritti
/home/administrator/Desktop/S	664
/home/administrator/Desktop/T	664
/home/administrator/Desktop/test/aaa.txt	444
/home/administrator/Desktop/test/b	775
/home/administrator/Desktop/test/CCCCC	664
/home/administrator/Desktop/test/DDDDDDDD	775
/home/administrator/Desktop/test/G.tzt	775
/home/administrator/Desktop/test/gg	664
/home/administrator/Desktop/test/KKK	664
/home/administrator/Desktop/test/zzzzZZzzz	667

Fig. 13: Lo stato dei file osservati come appare nel simulatore

3.3.5 Convertitore formato diritti

Per facilitare l'interpretazione dei diritti in forma ottale viene creata una funzione che associa ad ogni numero il corrispondente diritto[10]. Il tutto è realizzato tramite uno statement *switch*. La funzione prende come argomento un intero e restituisce una stringa.

```
<?php
function permessoUnix($dir) {
    switch($dir) {
     case 0:
         return "-";
         break;
     case 1:
         return "--x";
         break;
    ... ...        ...    ....
  }
}

?>
```

La stringa dell'ottale di tre cifre viene spezzata (es. *777* in 7,7,7) utilizzando la funzione `substr()` di php. Viene poi richiamata la funzione sopra per trasformare le cifre in stringhe.

```
<?php
$diritti = $_GET["d"];

$primo = substr($diritti,-3,1);    //estraggo il primo carattere
$secondo = substr($diritti,-2,1);  //estraggo il secondo..
$terzo = substr($diritti,-1);

echo "<p>" . "Forma alternativa: ";
echo "<b class=\"monos\">";
echo permessoUnix($primo) . permessoUnix($secondo) . permessoUnix($terzo);
echo "</b></p>";
....
?>
```

Ipotizzando una variabile *$GET*=777, l'output è "Forma alternativa: `rwxrwxrwx`".

[10] NERSC - `https://docs.nersc.gov/filesystems/unix-file-permissions/`

3.4 Utilizzo

3.4.1 Simulazione a partire da uno stato personalizzato

Nella pagina "simulatore" il programma permette all'utente di creare uno stato personalizzato inserendo il nome del file desiderato con relativi diritti[11] nel formato *xyz=123*. Il campo offre la funzione di auto completamento[12] attingendo alla base di dati contenente tutti i file osservati dal programma, tuttavia l'utente é anche libero di inserire dati propri.

Fig. 14: Simulazione di uno stato: creazione di uno stato personalizzato

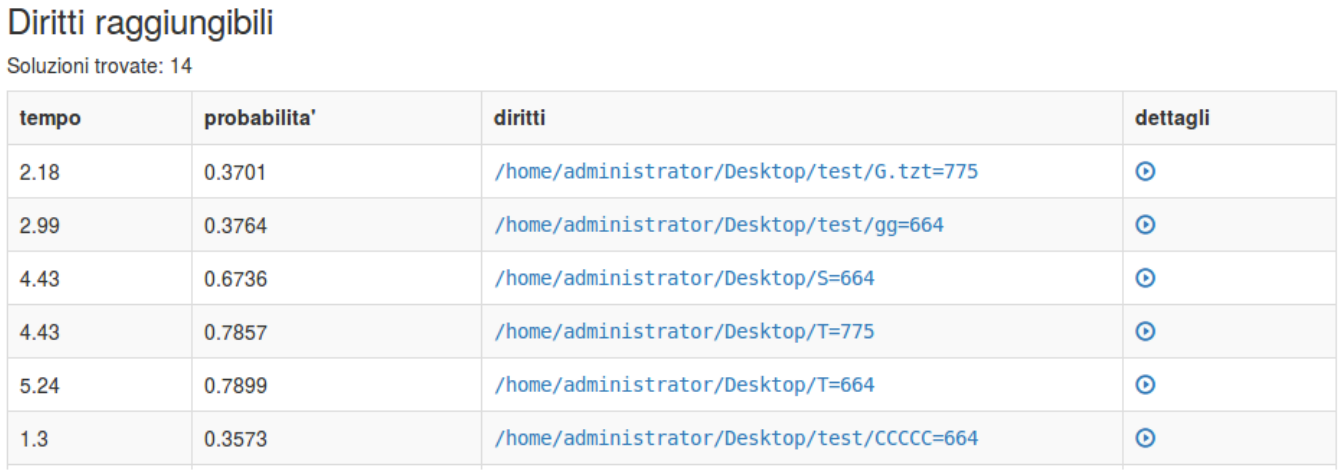
Premendo il pulsante *aggiungi* i diritti vengono aggiunti allo stato. L'opzione *nuovo stato* svuota lo stato. La computazione é istantanea durante l'aggiunta di nuovi diritti.

Fig. 15: Simulazione di uno stato: lista delle possibili vie per arrivare allo stato

Quando vengono individuati degli stati compatibili il programma ne mostra il numero, dopodiché come in **2.6.3** viene mostrato lo stato raggiungibile in tempo più breve e quello ottenibile più lentamente. Nell'ultima colonna vengono mostrati gli altri diritti acquisiti per arrivare a tale stato, inclusi i tempi in cui è possibile ottenerli a partire da *t=0*.

[11] nomeFile = diritti in formato ottale
[12] Tramite l'utilizzo di una richiesta asincrona XMLHttpRequest

Nella seconda parte dei risultati vengono mostrati tutti i diritti ottenibili a partire dallo stato trovato ottenuti utilizzando l'algoritmo *DFS* visto in **2.6.5**.



Diritti raggiungibili

Soluzioni trovate: 14

tempo	probabilita'	diritti	dettagli
2.18	0.3701	/home/administrator/Desktop/test/G.tzt=775	
2.99	0.3764	/home/administrator/Desktop/test/gg=664	
4.43	0.6736	/home/administrator/Desktop/S=664	
4.43	0.7857	/home/administrator/Desktop/T=775	
5.24	0.7899	/home/administrator/Desktop/T=664	
1.3	0.3573	/home/administrator/Desktop/test/CCCCC=664	

Fig. 16: Simulazione di uno stato: lista dei diritti raggiungibili dallo stato trovato

La soluzione mostrata è sempre la più veloce, tuttavia l'utente può richiamare una schermata dettagliata sulla raggiungibilità del diritto, premendo sull'icona nella colonna dettagli. Nell'esempio sotto la raggiungibilità di *G.tzt*.

Raggiungibilità **/home/administrator/Desktop/test/G.tzt=775**

Soluzioni trovate: 1

tempo	probabilita'	diritti acquisiti
2.18	0.3701	t=0.74 -> /home/administrator/Desktop/test/b=775 t=1.3 -> /home/administrator/Desktop/test/CCCCC=664 t=1.64 -> /home/administrator/Desktop/test/DDDDDDDD=775 t=2.18 -> /home/administrator/Desktop/test/G.tzt=775

Fig. 17: Simulazione di uno stato: selzionando un diritto raggiungibile è possibile consultare i dettagli sulla sua raggiungibilità

Nel caso i file in questione siano osservati dai sensori essi compaiono sotto forma di link cliccabile azzurro mostrando in una scheda separata la storia del file, inclusi gli eventi di apertura, chiusura e modifica degli attributi.[13]

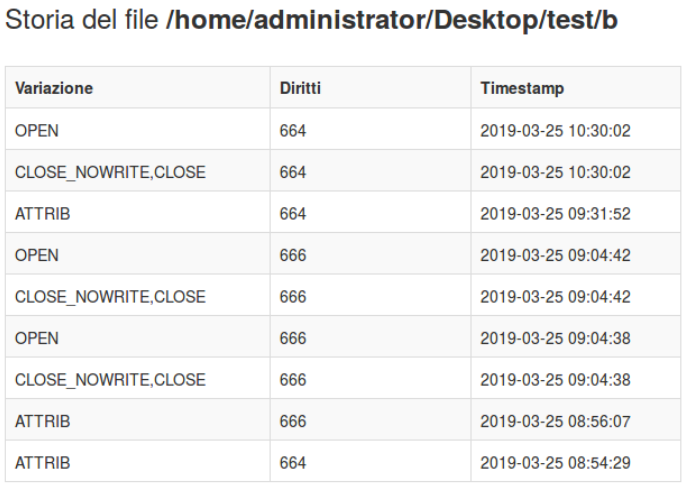


Fig. 18: Premendo su un sensore è possibile visualizzarne la storia

3.4.2 Simulazione a partire da uno stato reale

Nella pagina principale è possibile osservare la lettura dello stato attualmente osservato dai sensori attraverso il programma descritto in **2.7.2**. L'utilizzo è identico alla simulazione con stato personalizzato con l'unica differenza dettata dallo stato precompilato con i dati acquisiti dai sensori.

Sempre attraverso il pulsante *aggiungi* è possibile arrichire lo stato letto dai sensori con ulteriori diritti personalizzati. Premendo *nuovo stato* vengono eliminati dallo stato solamente i diritti aggiunti dall'utente ma non quelli letti dai sensori. La schermata dei risultati è identica a **3.3.1** mostrando all'utente quali sono i diritti raggiungibili e in quanto tempo.

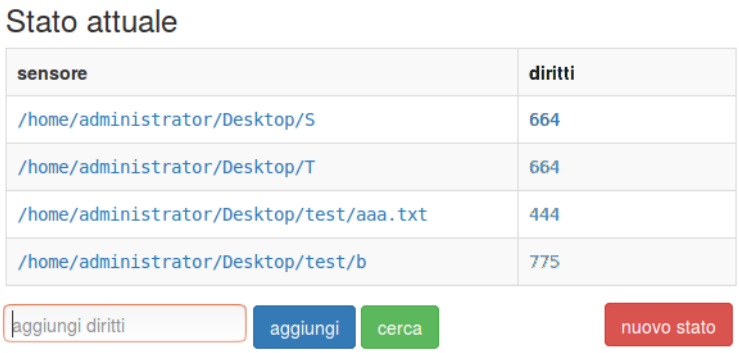


Fig. 19: Simulazione a partire da uno stato reale: viene visualizzato lo stato del sistema, l'utente può aggiungere ulteriori parametri alla simulazione

[13] Vedere *inotifytools* e la sezione **2.7**

3.4.3 Consultare i dati

Nella sezione *Consulta dati* è possibile visualizzare alcune statistiche relative ai dati e procedere nella navigazione per consultare i sensori attivi come in **3.3.1** o vedere i vari tipi di attacchi.

Fig. 20: La schermata visualizza alcune informazioni sui dati contenuti nel database

Premendo sugli attacchi catalogati viene aperta una scheda dedicata in cui è possibile consultare tutti gli attacchi presenti nel database, ordinati per tempo crescente.

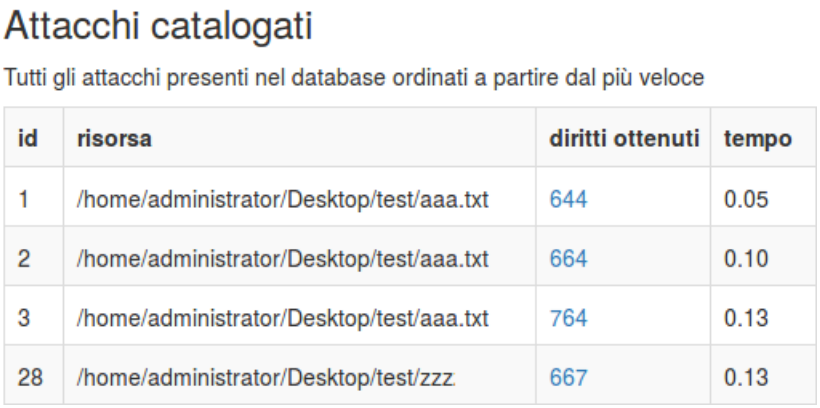
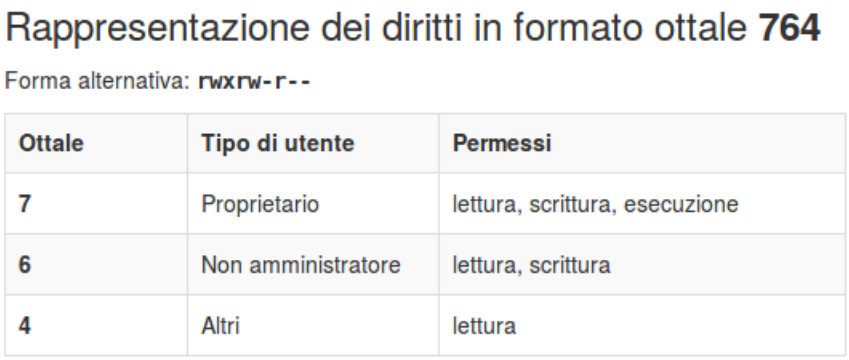


Fig. 21: Un riepilogo di tutti gli attacchi contenuti nel database

3.4.4 Rappresentazione dei diritti

Cliccando sui diritti in una qualunque pagina del programma viene aperta una scheda con utili informazioni riguardanti la codifica ottale dei diritti.



Rappresentazione dei diritti in formato ottale **764**

Forma alternativa: `rwxrw-r--`

Ottale	Tipo di utente	Permessi
7	Proprietario	lettura, scrittura, esecuzione
6	Non amministratore	lettura, scrittura
4	Altri	lettura

Fig. 22: Cliccando sui diritti appare questa schermata di approfondimento

4 Conclusioni

Questo progetto cerca di rispondere in maniera teorica alla raggiungibilità di uno stato e dei diritti raggiungibili da esso. Una implementazione attiva abbastanza veloce di questi algoritmi potrebbe permettere di sventare alcuni attacchi informatici, perlomeno quelli già conosciuti.
Oltre ad essere una soluzione specifica per il campo della sicurezza informatica, il programma tende anche ad essere una base generica per l'applicazione in altri campi.
La vastissima scala di applicabilità della teoria dei grafi permette poi, con poche modifiche, di adattare il tutto per lavorare su altri tipi di dati come reti informatiche o persino oggetti esterni all'informatica.
Inoltre il grafo orientato si presta bene anche per attività di linguistica computazionale, con l'algoritmo di raggiungibilità per esempio, è possibile costruire programmi in grado di gestire l'auto completamento dei caratteri durante la scrittura di una parola.
A prescindere dal tipo di applicazione, ovviamente è necessaria una grande quantità di dati a cui attingere per avere un risultato realistico e preciso.
Grazie ai big data infatti è possibile estrapolare informazioni spesso nascoste e legami tra esse che altrimenti sarebbero difficili da interpretare.

La visualizzazione dei dati da browser web è anch'essa fattore di indipendenza e scalabilità in qualsiasi ambiente. Permette all'utente di interfacciarsi con il software indipendentemente da dove si trova e dal dispositivo che utilizza, un ulteriore motivo per cui si rende necessaria una visualizzazione flessibile.
Con la divisione tra *server* e *client* l'utente può effettuare simulazioni pesanti eseguite sul server con milioni di vertici sul grafo in remoto dal proprio smartphone. In alternativa è possibile anche installare il software su più server, nei quali si voglia tenere sotto controllo lo stato del sistema e accedervi da un solo dispositivo.

5 Appendice

5.1 Bibliografia e strumenti utilizzati

- *Keijo Ruohonen*. Teoria dei grafi. Università di Tampere
  Scaricabile da: `http://math.tut.fi/~ruohonen/GT_English.pdf`
- *National Energy Research Scientific Computing*. Unix File Permissions
  Consultabile su: `https://docs.nersc.gov/filesystems/unix-file-permissions/`
- Documentazione ufficiale del linguaggio di programmazione *Python 3*
  Consultabile su: `https://docs.python.org/3/`
- Software per gestione di database relazionali *Oracle MySQL Workbench*
  Scaricabile da: `https://packages.debian.org/stretch/mysql-workbench`
- Bootstrap, libreria JavaScript per interfacce scalabili - Autore: Twitter
  Scaricabile da: `https://getbootstrap.com/`
- Framework *jQuery* - libreria JavaScript con licenza libera, creata da John Resig
  Ottenibile con documentazione presso: `https://jquery.com`
- Utility *inotifytools* - software open-source mantenuto da Radu Voicilas
  Scaricabile da: `https://packages.debian.org/stretch/inotify-tools`
  Documentazione ufficiale: `https://github.com/rvoicilas/inotify-tools/wiki`
- Tutte le grafiche sono realizzate dal candidato con l'uso dei software *draw.io* e *inkscape*
  Draw.io è utilizzabile presso: `https://draw.io`
  Inkscape è scaricabile da: `https://packages.debian.org/stretch/inkscape`
- Per la realizzazione di questo documento è stato utilizzato *Texmaker*
  Scaricabile da: `https://packages.debian.org/stretch/tex/texmaker`

5.2 Lista delle figure

5.3 Lista dei file

```
/var/www/
|   app.php
|   app_consulta_dati.php
|   app_impostazioni.php
|   app_nuovo_sensore.php
|   app_raggiungibilita.php
|   app_sensori.php
|   app_sensori_stats.php
|   app_simulazione.php
|   common_dash.php
|   common_end.php
|   common_head.php
|   script.js
|
+---backend
|   |   db_conn.php
|   |   get_dati_sensori.php
|   |   get_inserisci_cache_sim.php
|   |   get_lista_sensori.php
|   |   get_populate_request.php
|   |   get_python_script.php
|   |   get_ricerca_stato.php
|   |   get_storia_sensore.php
|   |   get_truncate_cache.php
|   |   get_suggerimento.php
|   |
|   \---python
|           ricerca_stato_reale.py
|           ricerca_simulatore.py
|           stato_attuale.py
|
+---css
|       bootstrap.min.css
|       jquery-ui.css
|       style.css
|
+---dati
|       grafo.json
|
+---fonts
|       glyphicons.ttf
|
+---lib
|   |   bootstrap-min.js
|   |   jquery-min.js
|   |   jquery-ui.js
|
\---sensori
        gestore_sensori
        monitor.py
        monitor.sh
```